Choix de noeud dans un algorithme de Branch and Bound sur intervalles

Bertrand Neveu, Gilles Trombettoni, Ignacio Araya

Choix de nœud dans un algorithme de *Branch and Bound* sur intervalles

Bertrand Neveu[1] **Gilles Trombettoni**[2] **Ignacio Araya**[3 *]

[1] Imagine LIGM, Université Paris–Est, France
[2] LIRMM, Université de Montpellier, CNRS, France
[3] Pontificia Universidad Católica de Valparaíso, Chile
Bertrand.Neveu@enpc.fr,Gilles.Trombettoni@lirmm.fr,rilianx@gmail.com

Résumé

Nous présentons dans cet article de nouvelles stratégies de choix de nœud dans un algorithme de *Branch and Bound* sur intervalles pour l'optimisation globale sous contraintes. La stratégie standard en meilleur d'abord choisit le nœud qui a la plus petite borne inférieure de l'estimation de l'objectif dans le domaine correspondant. Nous proposons d'abord de nouvelles stratégies qui prennent aussi en compte la borne supérieure de l'estimation de l'objectif. Une bonne précision sur cette borne supérieure, obtenue par l'application de plusieurs opérateurs de contraction, permet d'obtenir plus rapidement de bonnes solutions réalisables et donc de meilleures performances. Nous proposons également une autre stratégie qui réalise un compromis entre diversification et intensification en plongeant de manière gloutonne dans des régions potentiellement réalisables à chaque nœud de la recherche en meilleur d'abord. Ces nouvelles stratégies obtiennent de meilleurs résultats expérimentaux que la recherche en meilleur d'abord sur des instances difficiles d'optimisation globale sous contraintes.

Abstract

We present in this article new strategies for selecting nodes in interval Branch and Bound algorithms for constrained global optimization. The standard best-first strategy selects a node with the lowest lower bound of the objective estimate. We first propose new node selection policies where an *upper bound* of each node/box is also taken into account. The good accuracy of this upper bound achieved by several contracting operators leads to a good performance of the criterion. We propose another strategy that also makes a tradeoff between diversification and intensification by greedily diving into potential feasible regions at each node of the best-first search. These new strategies obtain better experimental results than classical best-first search on difficult constrained global optimization instances.

*Ignacio Araya est financé en partie par le projet Fondecyt 11121366.

1 Introduction

Cet article traite d'optimisation globale continue déterministe calculée par un algorithme de *Branch and Bound* sur intervalles. Plusieurs travaux ont été réalisés pour trouver de bonnes stratégies de branchement [10], mais la sélection de nœud elle-même a été peu étudiée. Les solveurs sur intervalles suivent généralement une stratégie en meilleur d'abord (BFS), et quelques études ont été menées pour limiter la croissance exponentielle de la mémoire utilisée [6, 18].

Travaux connexes

À notre connaissance, Casado et al. dans [6] et Csendes dans [9] ont été les seuls à proposer des stratégies de choix de nœud pour des algorithmes de B&B sur intervalles. Un critère à maximiser appelé C_3 dans [14] et adapté ensuite pour l'optimisation globale sans contraintes est :

$$\frac{f^* - lb}{ub - lb}$$

où $[lb, ub] = [f]_N([x])$ est l'intervalle obtenu par l'évaluation naturelle par intervalles de la fonction objectif f sur la boîte courante $[x]$: $[lb, ub]$ est un intervalle incluant tous les nombres réels images d'un point de $[x]$ par f. f^* est l'optimum. Quand f^* n'est pas connu, $\widetilde{f}$, le coût du meilleur point réalisable trouvé jusque là, peut être utilisé comme approximation de f^*. Ce critère favorise les petites boîtes et les nœuds avec une valeur de lb faible.

Pour l'optimisation sous contraintes, un autre critère (à maximiser) nommé C_5 est égal à $C_3 \times fr$. Il prend en compte un *ratio de faisabilité* fr calculé à partir de toutes les contraintes d'inégalité. Le critère C_7 propose de minimiser $\frac{lb}{C_5}$.

D'autres stratégies de choix de nœud ont été étudiées pour des algorithmes de B&B (non sur intervalles) en profondeur d'abord, en largeur d'abord ou en meilleur d'abord. La recherche en profondeur d'abord favorise l'exploitation (l'intensification). La recherche en meilleur d'abord favorise plus l'exploration (la diversification) car deux noeuds successifs peuvent appartenir à des régions significativement différentes de l'espace de recherche.

Différentes stratégies sont proposées dans l'optimiseur SCIP [20]. La profondeur, la largeur et le meilleur d'abord sont disponibles. Avec l'option `restartdfs`, SCIP réalise une recherche en profondeur d'abord, mais périodiquement (c.-à-d. tous les $k = 100$ retours en arrière), choisit le meilleur nœud. La stratégie la plus sophistiquée, appelée *Upper Confidence bounds for Trees* (UCT) [13], est utilisée pour les problèmes MIP.

La sélection de nœud de l'optimiseur global Baron est brièvement décrite dans [21]. La stratégie par défaut alterne entre le nœud ayant la violation minimum et celui de plus petite borne inférieure de l'estimation du coût. La violation d'un nœud est définie comme la somme des erreurs pour chaque variable, ces erreurs étant calculées à partir de la solution de la relaxation convexe du problème. Quand la limitation en mémoire devient critique, Baron passe en profondeur d'abord.

Des stratégies de choix de nœud ont aussi été définies pour des problèmes MINLP convexes. La stratégie par défaut de l'optimiseur `Bonmin` [5] choisit le nœud avec la plus petite borne inférieure de l'estimation du coût. Les stratégies en profondeur et en largeur d'abord sont aussi disponibles. Finalement, une stratégie dynamique commence en profondeur d'abord et passe en largeur d'abord après avoir trouvé trois solutions entières réalisables.

La recherche K-Best-First Search [11] s'applique aux problèmes d'optimisation combinatoire. KBFS(k) réalise de manière itérative des cycles d'expansion de nœuds. A chaque cycle, les k meilleurs nœuds sont traités et leurs enfants sont mis dans la liste des nœuds à traiter. Ainsi, KBFS réalise un compromis entre des recherches en largeur et en meilleur d'abord. Si $k = 1$, KBFS(1) mène une recherche en meilleur d'abord. Si $k = \infty$, alors tous les nœuds à un niveau sont traités avant de passer au niveau suivant ; KBFS(∞) mène donc une recherche en largeur d'abord.

Contribution et plan

Cet article propose de nouvelles stratégies pour des algorithmes de B&B sur intervalles.

Après une présentation des bases nécessaires à la compréhension de l'article dans la partie 2, une première approche est décrite dans la partie 3. Cette approche utilise, non seulement une borne inférieure mais aussi une borne supérieure de l'estimation de l'objectif sur un nœud. Cette borne supérieure est rendue plus précise en utilisant des méthodes de contraction qui éliminent des régions infaisables. La stratégie que nous proposons alterne entre le nœud avec la plus petite borne inférieure et celui avec la plus petite borne supérieure.

Une seconde approche réalise simplement un compromis entre exploration et exploitation. Un algorithme de B&B par intervalle en meilleur d'abord exécute à chaque nœud une plongée gloutonne en profondeur d'abord pour se focaliser sur des régions faisables (voir partie 4).

Dans la partie 5, une étude empirique des différentes variantes de ces nouvelles stratégies montre qu'elles obtiennent de bonnes performances sur des problèmes difficiles d'optimisation non convexe appartenant au banc d'essai Coconut.

2 Algorithmes de Branch and Bound sur intervalles

Un intervalle $[x_i] = [\underline{x_i}, \overline{x_i}]$ définit l'ensemble des nombres réels x_i tels que $\underline{x_i} \leq x_i \leq \overline{x_i}$. Une *boîte* $[x]$ est un produit cartésien d'intervalles $[x_1] \times ... \times [x_i] \times ... \times [x_n]$. $w([x])$ représente la taille du plus grand intervalle de la boîte $[x]$.

Cet article traite d'optimisation globale continue sous contraintes d'inégalités définie par :

$$\min_{x \in [x]} f(x) \quad s.c. \quad g(x) \leq 0$$

où $f : \mathbb{R}^n \to \mathbb{R}$ est la fonction objectif (non convexe) et $g : \mathbb{R}^n \to \mathbb{R}^m$ est un un vecteur de fonctions (non convexes) [1] $x = \{x_1, ..., x_i, ...x_n\}$ est un vecteur de variables dont le domaine est la boîte $[x]$. x est dit *réalisable* s'il satisfait les contraintes.

Un schéma de B&B sur intervalles pour l'optimisation globale continue sous contraintes (aussi connue comme programmation non linéaire NLP) est décrit ci-après. Les algorithmes 1 et 2 correspondent aux algorithmes implantés dans IbexOpt [22] et IBBA [19] et permettent de comprendre les stratégies de choix de nœud qui suivent. Par ailleurs, ces stratégies peuvent être incorporées dans tout algorithme de branch and bound sur intervalles.

L'algorithme de B&B maintient deux types principaux d'information durant la recherche : $\tilde{f}$, le coût du meilleur point réalisable trouvé jusque là et f_{min} la plus petite borne inférieure $[x].lb$ de l'estimation de l'objectif sur les nœuds $[x]$ à explorer. En d'autres termes, pour chaque boîte $[x]$, il est garanti qu'il n'existe pas de point réalisable avec un coût inférieur à $[x].lb$.

1. Les équations $h_j(x) = 0$ peuvent être relâchées en deux inégalités $-\epsilon \leq h_j(x) \leq +\epsilon$.

Algorithm `IntervalBranch&Bound` *(f, g, x, box, ϵ_{obj}, ϵ_{sol})*

$f_{min} \leftarrow -\infty$; $\widetilde{f} \leftarrow +\infty$; $x_{\widetilde{f}} \leftarrow \bot$
$f^{sb}_{min} \leftarrow +\infty$ /* La plus petite borne inférieure des nœuds ayant atteint la précision ϵ_{sol}. */
$Boxes \leftarrow \{box\}$
while $Boxes \neq \emptyset$ **and** $\widetilde{f} - f_{min} > \epsilon_{obj}$ **and** $\frac{\widetilde{f}-f_{min}}{|\widetilde{f}|} > \epsilon_{obj}$ **do**
  $[x] \leftarrow$ `BestBox` ($Boxes$, criterion) ; $Boxes \leftarrow Boxes \setminus \{[x]\}$
  $([x]_1, [x]_2) \leftarrow$ `Bisect` ($[x]$)
  $([x]_1, x_{\widetilde{f}}, \widetilde{f}) \leftarrow$ `Contract&Bound` ($[x]_1$, f, g, x, ϵ_{obj}, $x_{\widetilde{f}}$, $\widetilde{f}$)
  $([x]_2, x_{\widetilde{f}}, \widetilde{f}) \leftarrow$ `Contract&Bound` ($[x]_2$, f, g, x, ϵ_{obj}, $x_{\widetilde{f}}$, $\widetilde{f}$)
  $(f^{sb}_{min}, Boxes) \leftarrow$ `UpdateBoxes` ($[x]_1$,ϵ_{sol}, f^{sb}_{min},$Boxes$)
  $(f^{sb}_{min}, Boxes) \leftarrow$ `UpdateBoxes` ($[x]_2$,ϵ_{sol}, f^{sb}_{min}, $Boxes$)
  $f_{min} \leftarrow \min(f^{sb}_{min}, \min_{[x]\in Boxes}\ [x].lb)$

Algorithm 1: Branch and Bound sur intervalles

L'algorithme est lancé avec l'ensemble de contraintes g, la fonction objectif f et avec la boîte des domaines d'entrée initialisant la liste $Boxes$ des boîtes à traiter. ϵ_{obj} est la précision requise sur l'objectif et est utilisé comme critère d'arrêt. Nous ajoutons au système une variable x_{obj} correspondant à l'image (au coût) de la fonction objectif et une contrainte $f(x) = x_{obj}$.

La procédure `BestBox` choisit le prochain nœud à traiter. Si le critère choisit un nœud $[x]$ avec la plus petite borne inférieure de l'estimation de l'objectif ($[x].lb$), l'algorithme B&B suivra la stratégie la plus commune (le meilleur d'abord).

La boîte choisie est ensuite coupée en deux sous-boîtes $[x]_1$ et $[x]_2$ le long d'une dimension choisie par une stratégie de branchement.

Les deux sous-boîtes sont alors traitées par la procédure `Contract&Bound` (voir l'algorithme 2). Une contrainte $x_{obj} \leq \widetilde{f} - \epsilon_{obj}$ est ajoutée au système pour diminuer la borne supérieure de l'estimation de l'objectif dans la boîte. La valeur ϵ_{obj} permet que la prochaine solution trouvée soit significativement meilleure que le meilleur point réalisable courant. La procédure `Contraction` *contracte* ensuite la boîte courante sans perdre de partie réalisable. En d'autres termes, quelques parties non réalisables aux bords du domaine sont enlevées par des algorithmes issus de la programmation par contraintes (CP) et des algorithmes de convexification. Cette contraction travaille sur la boîte étendue incluant la variable objectif x_{obj}. Ainsi, augmenter $\underline{x_{obj}}$ revient à augmenter la borne inférieure de l'objectif sur la boîte courante.

La procédure `FeasibleSearch` appelle ensuite une ou plusieurs heuristiques recherchant un point réalisable $x_{\widetilde{f}}$ qui améliore le meilleur coût trouvé jusqu'à présent (*upperbounding*).

Les derniers appels à `UpdateBoxes`, décrits à l'algorithme 3, mettent généralement les deux sous-boîtes dans l'ensemble $Boxes$ des nœuds à explorer. Si la taille d'une boîte atteint la précision ϵ_{sol}, cette boîte ne sera pas étudiée (bissectée) et seule la valeur f^{sb}_{min} sera mise à jour.

On remarquera que si l'arbre de recherche est parcouru en meilleur d'abord, une mémoire de taille exponentielle peut être requise pour stocker les nœuds à traiter.

Rappelons que $[x].lb$ est un minorant de l'objectif dans la boîte $[x]$ en tenant compte des contraintes (il n'existe aucun point réalisable avec un coût inférieur à $[x].lb$) et $[x].ub$ est un majorant de l'objectif pour le prochain point réalisable recherché dans la boîte courante.

L'arithmétique par intervalles appliquée sur l'objectif donne facilement des premières valeurs pour ces bornes. Avec l'arithmétique par intervalles, on remplace les opérateurs mathématiques standard dans f par leurs équivalents sur intervalles pour calculer $[f]([x])$. Ainsi, $\underline{[f]([x])}$ donne une valeur pour $[x].lb$ tandis que $\overline{[f]([x])}$ calcule une valeur pour $[x].ub$ Nous

Algorithm `Contract&Bound` *($[x]$,f,g,x,ϵ_{obj},$x_{\widetilde{f}}$,$\widetilde{f}$)*

$g' \leftarrow g \cup \{x_{obj} \leq \widetilde{f} - \epsilon_{obj}\}$
$[x] \leftarrow$ `Contraction` ($[x]$, $g' \cup \{f(x) = x_{obj}\}$)
if $[x] \neq \emptyset$ **then**
  /* Upperbounding */
  $(x_{\widetilde{f}}$,cost)$\leftarrow$`FeasibleSearch` ($[x]$,f,g',ϵ_{obj})
  if $cost < \widetilde{f}$ **then** $\widetilde{f} \leftarrow$ cost
return ($[x]$, $x_{\widetilde{f}}$, $\widetilde{f}$)

Algorithm 2: La procédure Contract&Bound lancée à chaque nœud de l'algorithme B&B

```
Algorithm UpdateBoxes ([x], ε_sol, f^sb_min, Boxes)
    if [x] ≠ ∅ then
        if w([x]) > ε_sol then
            Push ([x], Boxes)
        else
            f^sb_min ← min(f^sb_min, x_obj)
            /* or f^sb_min ← min(f^sb_min, [f]([x])) */
    return (f^sb_min, Boxes)
```

Algorithm 3: Procédure de mise à jour des listes de nœuds à explorer ou du coût minimum de l'objectif sur les petites boites qui ne seront pas étudiées

verrons dans la section suivante comment ces bornes sont affinées en tenant compte des contraintes, donc de l'espace réalisable, et de la précision ϵ_{obj}.

3 Nouvelles stratégies basées sur le calcul de bornes supérieures

En optimisation, le choix du prochain nœud à traiter est crucial pour obtenir de bonnes performances. Si la stratégie habituelle de choix du nœud ayant la plus petite borne inférieure de l'estimation de l'objectif (stratégie en meilleur d'abord) permet une recherche optimale en optimisation sans contraintes, ce n'est plus le cas avec des contraintes.

Il est aussi intéressant d'obtenir rapidement de bons points réalisables, pour limiter d'une part la taille de la liste des nœuds à traiter et pour réduire l'espace de recherche en propageant les contraintes. Le meilleur nœud est alors celui qui améliore le plus le meilleur point réalisable. En effet, cette amélioration de la borne supérieure réduit *globalement* l'espace réalisable grâce à la contrainte $x_{obj} \leq \widetilde{f}$. Il existe deux phases dans un B&B :

1. une phase où on essaye de trouver la solution optimale,
2. une seconde phase où on prouve que cette solution est optimale, ce qui requiert de traiter tous les nœuds restants.

Le choix de nœud ne concerne donc que la première phase.

Nous définissons dans cette section de nouvelles stratégies agrégeant deux critères pour choisir la boîte courante :

1. LB : Le critère bien connu utilisé en meilleur d'abord choissant le nœud ayant le plus petit $[x].lb$. Ce critère est optimiste puisque nous espérons trouver une solution avec un coût f_{min}, auquel cas la recherche est terminée.
2. UB : Ce critère choisit le nœud ayant la plus petite borne supérieure de l'estimation de l'objectif, le nœud avec le plus petit $[x].ub$. Ainsi, si un point réalisable est trouvé, on espère qu'il améliorera de manière importante le meilleur coût trouvé jusque là.

Le critère UB est le symétrique du critère LB. Pour chaque boîte, $[x].lb$ et $[x].ub$ sont calculés par la procédure `Contract&Bound` et étiquettent le nœud avant de le stocker dans l'ensemble des nœuds à traiter. Rappelons qu'une variable x_{obj} représentant la valeur de l'objectif a été introduite. Ainsi, $[x].lb$ et $[x].lb$ sont simplement les bornes de l'intervalle $[x_{obj}]$ après contraction. Des techniques de programmation par contraintes comme 3BCID [23] peuvent améliorer $[x_{obj}]$ en traitant et enlevant de petites tranches aux bornes de $[x_{obj}]$ (rognage).

Nous pensons qu'une clé du succès du critère UB est qu'il évalue l'objectif plus précisément que l'évaluation naturelle ne le ferait (ub en calculant $[lb, ub] \leftarrow [f]([x])$).

Nous proposons deux manières d'agréger ces deux critères.

Sommer les deux critères : `LB+UB`. Cette stratégie choisit le nœud avec la plus petite valeur de la somme $[x].lb+[x].ub$. Cela correspond à minimiser les deux critères avec le même poids, c.-à-d. minimiser le milieu de l'intervalle représentant l'estimation de l'objectif dans la boîte.

Alterner les deux critères : `LBvUB`. Dans cette deuxième stratégie, on choisit la prochaine boîte à traiter en utilisant *un* des deux critères. Un choix aléatoire est effectué par la fonction *BestBox* à chaque selection de nœud, avec une probabilité `UBProb` de choisir UB. Si UB (resp. LB) est choisi et si plusieurs nœuds ont le même coût $[x].ub$ (resp. $[x].lb$), alors nous utilisons l'autre critère LB (resp. UB) pour départager les exaequo.

Des expérimentations ont montré que les performances n'étaient pas sensibles à un réglage fin du paramètre `UBProb` pourvu qu'il reste entre 0.2 et 0.8 ; ainsi il a été fixé à 0.5. Les expérimentations de la partie 5 soulignent l'impact positif de ce critère sur les performances.

3.1 Améliorer le critère UB en favorisant les régions réalisables

Tous les nœuds à explorer ont une étiquette UB dépendant du meilleur coût trouvé au moment où les boîtes ont été traitées par `Contract&Bound`. Grâce aux modifications apportées à `Contract&Bound` (voir l'algorithme 4), ces étiquettes tombent dans l'une des quatre catégories de coût dépendant de la valeur $\widetilde{f}$.

Algorithm `Contract&Bound` $([x], f, g, x, \epsilon_{obj}, x_{\widetilde{f}}, \widetilde{f})$

$g' \leftarrow g \cup \{x_{obj} \leq \widetilde{f} - (\epsilon_{obj} - 0.1\epsilon_{obj})\}$
$[x] \leftarrow$ `Contraction` $([x], g' \cup \{f(x) = x_{obj}\})$
if $[x] \neq \emptyset$ **then**
  // Upperbounding :
  $(x_{\widetilde{f}}, \text{cost}) \leftarrow$ `FeasibleSearch` $([x], f, g', \epsilon_{obj})$
  if $cost < \widetilde{f}$ **then**
    $\widetilde{f} \leftarrow \text{cost}$
    $[x].ub \leftarrow \widetilde{f} - \epsilon_{obj}$
  else
    $[x].ub \leftarrow \overline{x_{obj}}$
return $([x], x_{\widetilde{f}}, \widetilde{f})$

Algorithm 4: Modification de la procédure `Contract&Bound` pour améliorer le critère UB

Elles indiquent avec quelle priorité les boîtes sont choisies quand le critère UB est utilisé. L'étiquette UB $[x].ub$ est :

1. inférieure à $\widetilde{f} - \epsilon_{obj}$ si la procédure de contraction a réduit le maximum de l'estimation de l'objectif dans la boîte,
2. égale à $\widetilde{f} - \epsilon_{obj}$, si la boîte est un descendant de la boîte contenant le meilleur point réalisable actuel de coût $\widetilde{f}$,
3. égale à $\widetilde{f} - 0.9\,\epsilon_{obj}$ si la boîte a été traitée après la dernière mise à jour de $\widetilde{f}$,
4. supérieure à $\widetilde{f} - 0.9\,\epsilon_{obj}$ dans le cas restant.

Comme on le voit dans le pseudo-code de `Contract&Bound`, le terme additionnel $0.1\,\epsilon_{obj}$ pénalise les boîtes qui ne sont pas issues par bissections successives de la boîte où le meilleur point réalisable actuel a été trouvé.

Les bons résultats expérimentaux obtenus par la stratégie `LBvUB` (cf. partie 5) suggèrent qu'il est important de réaliser à la fois une intensification (UB) et une diversification (LB) de la recherche. L'ajout d'un second critère permet d'éviter les inconvénients de l'utilisation d'un seul critère, c.-à-d. (pour LB) choisir des boîtes prometteuses sans point réalisable et (pour UB modifié) aller plus en profondeur dans l'arbre de recherche en restant dans un minimum local.

3.2 Implantation de l'ensemble des nœuds à explorer

Dans `IbexOpt`, l'ensemble *Boxes* des nœuds à explorer était initialement implanté par une structure de tas ordonné par le critère LB. Pour la stratégie `LBvUB`, chaque nœud dans l'ensemble *Boxes* est étiqueté par deux valeurs : $[x].lb$ et $[x].ub$ et on a essayé plusieurs implantations pour trier les nœuds selon les deux critères LB et UB.

1. *Un tas* ordonné par $[x].lb$:
   Le choix de nœud utilisant UB est alors effectué en un coût linéaire par rapport au nombre de nœuds à explorer. En pratique, le temps nécessaire à la gestion du tas représente environ 10% du temps de calcul total quand ce nombre de nœuds dépasse 50000.
2. Une variante utilisant toujours un seul tas teste la taille du tas $|Boxes|$:
   Si $|Boxes|$ dépasse 50000, la probabilité `UBProb` est mise à 0.1 ; sinon, `UBProb` reste égal à 0.5. Cette variante produit donc moins d'appels avec le critère UB.
3. *Deux tas* :
   Finalement, nous avons essayé une implantation avec deux tas, l'un pour LB, l'autre pour UB. Toutes les opérations sont alors en $O(log_2(|Boxes|))$, sauf le processus de filtrage du tas lancé quand un nouveau point réalisable est trouvé. Ce ramasse-miettes enlève de *Boxes* tous les nœuds ayant $[x].lb > \widetilde{f}$.

4 La stratégie "Feasible Diving"

Nous avons conçu une autre stratégie de choix de nœud qui réalise aussi un compromis entre exploitation et exploration. L'algorithme B&B par intervalles réalise une recherche en meilleur d'abord décrite à l'algorithme 5.

Cette stratégie utilise une variante du critère LB : elle choisit un nœud avec le plus petit $[x].lb$, mais nous avons juste ajouté deux autres critères pour départager les ex-aequo :

1. le nœud le plus haut dans l'arbre de recherche.
2. en cas d'égalité sur $[x].lb$ et sur ce critère, on choisit le nœud généré en premier.

À chaque nœud de cette recherche, nous appelons une procédure effectuant une recherche gloutonne en profondeur d'abord pour intensifier la recherche dans la région de la boîte courante. Cette procédure *Feasible Diving* est décrite à l'algorithme 6. À partir du nœud sélectionné, `FeasibleDiving` construit un arbre en profondeur et garde le nœud le plus prometteur à chaque itération. La boîte est bissectée sur une dimension choisie par la même stratégie que dans l'algorithme B&B, par exemple *SmearSum relative* [22] et les deux sous-boîtes sont traitées par la procédure `Contract&Bound` (voir l'algorithme 2). La sous-boîte avec la plus petite borne inférieure de l'objectif est traitée par l'itération suivante de *Feasible Diving* tandis que l'autre est mise dans le tas global *Boxes* stockant la liste des nœuds à explorer.

Algorithm `IntervalBranch&BoundBis` *(f, g, x, box, ϵ_{obj}, ϵ_{sol})*

$f_{min} \leftarrow -\infty$; $\widetilde{f} \leftarrow +\infty$; $x_{\widetilde{f}} \leftarrow \bot$
$f^{sb}_{min} \leftarrow +\infty$ /* La plus petite borne inférieure des nœuds ayant atteint la précision ϵ_{sol}. */
$Boxes \leftarrow \{box\}$
while $Boxes \neq \emptyset$ **and** $\widetilde{f} - f_{min} > \epsilon_{obj}$ **and** $\frac{\widetilde{f}-f_{min}}{|\widetilde{f}|} > \epsilon_{obj}$ **do**
  $[x] \leftarrow$ BestBox ($Boxes$, criterion) ; $Boxes \leftarrow Boxes \setminus \{[x]\}$
  $([x], x_{\widetilde{f}}, \widetilde{f}, f^{sb}_{min}, Boxes) \leftarrow$ `FeasibleDiving` ($[x]$, g, f, x, ϵ_{sol}, $x_{\widetilde{f}}$, $\widetilde{f}$, f^{sb}_{min}, $Boxes$)
  if $[x] \neq \emptyset$ /* $w([x]) < \epsilon_{sol}$ */ **then** $f^{sb}_{min} \leftarrow min(f^{sb}_{min}, \underline{x_{obj}})$
  $f_{min} \leftarrow \min(f^{sb}_{min}, \min_{[x]\in Boxes} [x].lb)$

Algorithm 5: Branch and Bound sur intervalles (variante appelant Feasible Diving)

Algorithm `FeasibleDiving` *($[x]$, g, f, x, ϵ_{sol}, $x_{\widetilde{f}}$, $\widetilde{f}$, f^{sb}_{min}, $Boxes$)*

while $[x] \neq \emptyset$ **and** $w([x]) > \epsilon_{sol}$ **do**
  $([x]_1, [x]_2) \leftarrow$ `Bisect` ($[x]$)
  $([x]_1, x_{\widetilde{f}}, \widetilde{f}) \leftarrow$ `Contract&Bound` ($[x]_1$, f, g, x, ϵ_{obj}, $x_{\widetilde{f}}$, $\widetilde{f}$)
  $([x]_2, x_{\widetilde{f}}, \widetilde{f}) \leftarrow$ `Contract&Bound` ($[x]_2$, f, g, x, ϵ_{obj}, $x_{\widetilde{f}}$, $\widetilde{f}$)
  $([x]_{best}, [x]_{worst}) \leftarrow$ `Sort` ($[x]_1$, $[x]_2$, *criterion*) /* $[x]_{best}.lb < [x]_{worst}.lb$ */
  $(f^{sb}_{min}, Boxes) \leftarrow$ `UpdateBoxes` ($[x]_{worst}$,ϵ_{sol}, f^{sb}_{min}, $Boxes$)
  $[x] \leftarrow [x]_{best}$
return ($[x]$, $x_{\widetilde{f}}$, $\widetilde{f}$, f^{sb}_{min}, $Boxes$)

Algorithm 6: La procédure `Feasible Diving` lancée à chaque nœud du B&B en meilleur d'abord.

La procédure `FeasibleDiving` essaye de plonger dans une région réalisable (ou de montrer que la région explorée ne contient pas de point réalisable) grâce aux procédures `Contraction` et `FeasibleSearch` appelée à chaque itération. Avec cette plongée, on espère que ces procédures trouveront des points réalisables dans des petites sous-boîtes.

Il est mentionné dans [5] et [4] qu'une procédure similaire a été proposée pour des solveurs MIP. Bien que non détaillée dans la littérature, cette procédure *Probed Diving* semble être utilisée par le solveur MIP CPLEX d'IBM Ilog avec un certain succès.

5 Expérimentations

Nous avons mené des expérimentations sur un ensemble de problèmes d'optimisation globale continue sous contraintes. La partie 5.1 décrit le protocole expérimental. La partie 5.2 détaille les essais réalisés avec les différentes stratégies mixant LB et UB, spécialement `LB+UB` et `LBvUB`. Les parties 5.3 et 5.4 comparent `FeasibleDiving` à la stratégie standard en meilleur d'abord et à `LBvUB`.

5.1 Protocole expérimental et implantation

Toutes les variantes de notre solveur NLP `IbexOpt` [22] ont été implantées dans la bibliothèque libre en `C++` Interval-Based EXplorer (`Ibex`) [8, 7]. Les principaux composants de `IbexOpt` sont :

- Pour la contraction et l'estimation de la borne inférieure :
  - des contracteurs provenant de la programmation par contraintes sur intervalles comme l'algorithme bien connu de propagation de contraintes HC4 [15, 3] et le contracteur plus récent ACID [17].
  - des contracteurs réalisant une relaxation linéaire des contraintes et de l'objectif `X-Taylor` [1] et `ART` [16] (relaxation basée sur l'arithmétique affine).
- Pour la recherche d'un point réalisable :
  Une approche originale extrait une *région intérieure* (boîte ou polytope) dans l'espace réalisable en utilisant les algorithmes `InHC4` et `InXTaylor` [2].

On notera qu'aucune méthode lagrangienne ou d'analyse convexe n'est utilisée dans la version courante d'`IbexOpt`.

Nous avons sélectionné toutes les instances des séries 1 et 2 du banc d'essai sur l'optimisation globale sous contraintes Coconut [2] qui :

— sont résolues en un temps compris entre 1 seconde et 1 heure par un compétiteur (précision ϵ_{obj} = `1e-8`),
— possèdent entre 6 et 50 variables,
— sont résolues par `IbexOpt` en utilisant la meilleure stratégie de branchement parmi *Smear sum* absolue (ssa) ou relative (ssr), *Smear max* (sm), *Intervalle le plus large* (lf) et *Tour de rôle* (rr).

Cela donne les 84 instances listées en annexe (voir tableau 6).

5.2 Tests sur des stratégies mixtes LB/UB

Les expérimentations sur les meilleures stratégies mélangeant les critères LB et UB sont décrites dans cette partie. Par rapport à la stratégie LB standard (en meilleur d'abord), la meilleure stratégie de choix de nœud obtient un gain d'environ 40% sur le temps total et de 25% en moyenne, ce qui signifie que les plus grands gains sont obtenus sur les problèmes difficiles. De plus, on obtient un gain significatif sur plusieurs instances et la stratégie est robuste, c.-à-d. aucune perte importante par rapport à LB n'a été observée.

Comparaison entre LB+UB et les variantes LBvUB

Le tableau 1 montre les résultats expérimentaux obtenus avec différentes variantes mixant les critères LB et UB. Les gains/pertes par rapport à LB sont exprimés par $\frac{time(LB)}{time(LB/UB)}$.

Critère	#tas	t(s) max perte	t(s) max gain	t(s) moy. gain	t(s) total gain	nds moy. gain	nds total gain
LB+UB	1	0.14	10.2	1.21	1.54	1.28	1.70
LBvUB	1	0.52	9.17	1.17	1.58	1.20	1.75
LBvUB-01	1	0.52	21.7	1.18	1.64	1.19	1.67
LBvUB	2	0.46	21.7	1.28	1.67	1.26	1.83

TABLE 1 – Comparaison entre `LB+UB` et les variantes `LBvUB`. `LBvUB-01` est `LBvUB` avec la probabilité *UBProb* égale à 0.1 quand la taille du tas dépasse 50000 nœuds.

D'après ces expérimentations, LBvUB avec deux tas (un pour chaque critère) semble être la meilleure variante et l'expérimentation suivante justifie le choix de cette variante pour le critère UB.

2. `www.mat.univie.ac.at/~neum/glopt/coconut/Benchmark/Benchmark.html`

Quel est le meilleur critère UB ?

Nous avons réalisé d'autres tests sur ces 84 instances avec une précision sur l'objectif ϵ_{obj} = `1.e-6` (au lieu de `1.e-8`) en utilisant la version plus récente 2.1.10 d'`Ibex`. Toutes ces expérimentations utilisent une stratégie "LBvAutre" (avec une probabilité 0.5 de choisir un des deux critères pour le choix de nœud). Le premier critère est LB et l' "autre" critère est :

— UB_0 : UB sans biais favorisant les descendants (sous-boîtes) dans l'arbre du B&B du nœud où le dernier $\widetilde{f}$ a été trouvé.
— UB_1 : UB avec la modification de la procédure `Contract&Bound` décrite dans l'algorithme 4 qui favorise les descendants du nœud où le dernier $\widetilde{f}$ a été trouvé.
— FUB : UB avec un biais plus fort pour les descendants des nœuds où un point réalisable a été trouvé. Un tel descendant avec le plus petit UB est choisi, si il existe ; dans le cas contraire, on choisit un nœud avec le plus petit UB.
— MID : valeur de la fonction objectif au milieu de la boîte, donc en oubliant la région réalisable.
— C_3, C_5 ou C_7 : les critères décrits dans [14] et mentionnés dans l'introduction.

Le tableau 2 montre une comparaison synthétique de toutes ces variantes. Nous utilisons une formule plus sophistiquée pour les gains, qui donne une mesure plus équitable. La valeur dans chaque case correspond à une moyenne d'un ratio de temps normalisé entre une stratégie s et la stratégie de référence LB. Une valeur inférieure à 1 correspond à un gain, une valeur supérieure à 1 une perte. La mesure prend en compte le temps CPU d'une stratégie s pour résoudre une instance i (ce temps CPU est en fait une moyenne sur 10 essais effectués avec différentes graines pour les choix aléatoires effectué dans IbexOpt). Il est calculé de la manière suivante :

— $timeRatio(i,s) = \frac{time(i,s)}{time(i,s)+time(LB,i)}$
— $averageTimeRatio(s)$ est la moyenne des $timeRatio(i,s)$.
— ratio de temps normalisé (valeur pour les gains de s) = $\frac{averageTimeRatio(s)}{1-averageTimeRatio(s)}$

	UB_0	UB_1	FUB	MID	C_3	C_5	C_7
sur les 84 instances	0.87	0.84	0.87	0.91	0.93	1.03	1.08
sur les inst. > 10 s.	0.80	0.74	0.77	0.88	0.87	0.94	0.99

TABLE 2 – Comparaison entre les stratégies de choix de nœud `LB` et `LBvAutre` (`Autre` définissant la colonne)

La stratégie `LBvUB`, avec la variante UB_1 présentée à l'algorithme 4, obtient les meilleurs résultats. On observe un gain de 26% de UB_1 sur LB sur les problèmes difficiles.

Les critères C_3, C_5 et C_7 ont donné de moins bons résultats sur notre échantillon. Cela suggère qu'une évaluation précise de $[x].lb$ et de $[x].ub$ obtenue par contraction est en pratique meilleure qu'une évaluation par l'arithmétique d'intervalles de ces valeurs et que le *ratio de faisabilité* n'est peut être pas une mesure pertinente.

Comparaison détaillée entre LB et LBvUB

La figure 1 montre un diagramme comparant la meilleure variante de `LBvUB` ($\texttt{LBvUB}_1$) avec LB. Le tableau 3 confirme les bénéfices de LBvUB.

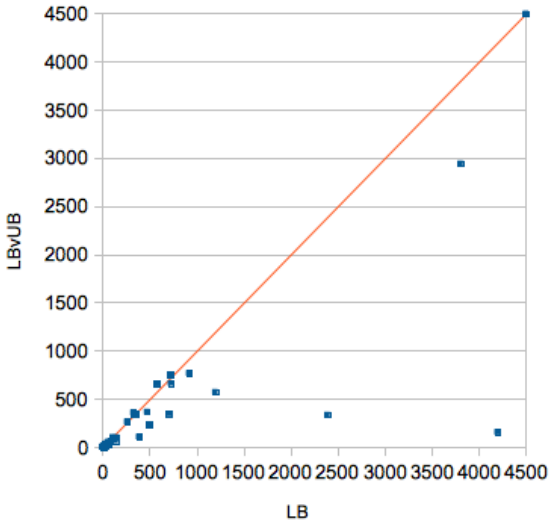

FIGURE 1 – `LB` versus `LBvUB`. Les coordonnées de chaque point représentent le temps CPU (en secondes) utilisé par les compétiteurs. La plupart des points sont sous la diagonale, ce qui illustre le fait que LBvUB domine LB.

5.3 Comparison entre LB et FeasibleDiving

La figure 1 montre un diagramme comparant `FeasibleDiving` avec LB. Le tableau 3 confirme les bénéfices de `FeasibleDiving`.

5.4 Comparaison entre LBvUB et FeasibleDiving

Nous concluons cette partie expérimentale en comparant les deux stratégies présentées dans cet article.

La figure 3 montre un diagramme comparant la meilleure variante de `LBvUB` ($\texttt{LBvUB}_1$) à FD. Le tableau 5 confirme les bénéfices de `FeasibleDiving` qui permet de résoudre les deux instances `concon` et `mconcon`.

Remarque : rappelons que l'algorithme B&B appelle une procédure de recherche de meilleur point réalisable à chaque nœud exploré par `FeasibleDiving`.

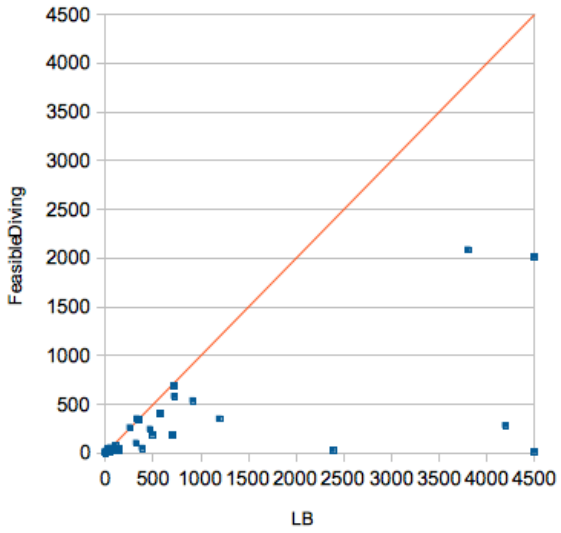

FIGURE 2 – LB versus FD. Les coordonnées de chaque point représentent le temps CPU (en secondes) utilisés par les compétiteurs. La plupart des points apparaissent sous la diagonale ce qui montre que FD domine nettement LB.

En effet, chaque itération de `FeasibleDiving` appelle `Contract&Bound` qui lui-même appelle les procédures `Contraction` et `FeasibleSearch`. Il est possible d'appeler si souvent `FeasibleSearch` dans `IbexOpt` parce que cela correspond à deux procédures relativement peu coûteuses (`InHC4` et `In-XTaylor` dans[2]). Quand l'algorithme plonge, la boîte courante devient petite et est davantage susceptible soit de contenir un espace réalisable que ces procédures arrivent à trouver, soit d'être éliminée par les procédures de contraction. Ceci pourrait expliquer les bons résultats obtenus par `FeasibleDiving`.

6 Conclusion

La stratégie de choix de nœud est une voie de recherche prometteuse pour améliorer la performance des algorithmes de B&B sur intervalles. Nous avons proposé dans cet article deux nouvelles stratégies qui obtiennent de bons résultats expérimentaux sur des instances difficiles.

Ces deux approches sont basées sur un algorithme de recherche en meilleur d'abord. La première stratégie, appelée `LBvUB`, alterne entre la sélection d'un nœud avec la plus petite borne inférieure (LB) de l'objectif et d'un nœud avec la plus petite borne supérieure (UB). Le critère UB est biaisé pour favoriser les descendants du nœud où le dernier point réalisable a été trouvé. Les bornes inférieures et supérieures sont rendues plus précises par des opérations de contraction.

La seconde stratégie de choix de nœud, `Feasible-`

Gain	gain >5	gain [2,5]	gain [1.2,2]	gain [1.05,1.2]	équiv. [0.95,1.05]	perte [0.8,0.95]	perte [0.5,0.8]	perte <0.5
#instances	4	7	29	21	15	5	0	1

TABLE 3 – Détail des gains et pertes de `LBvUB` par rapport à LB. Les gains sont exprimés par $\frac{time(LB)}{time(LBvUB)}$. Les valeurs indiquent le nombre d'instances avec un gain dans un intervalle donné.

Gain	gain > 5	gain $[2, 5]$	gain $[1.2, 2]$	gain $[1.05, 1.2]$	équiv. $[0.95, 1.05]$	perte $[0.8, 0.95]$	perte $[0.5, 0.8]$	perte < 0.5
#instances	4	21	21	9	18	9	2	0

TABLE 4 – Détail des gains/pertes de FD par rapport à LB. Les gains sont exprimés par $\frac{time(LB)}{time(FD)}$. Les valeurs donnent le nombre d'instances ayant un gain dans un intervalle donné.

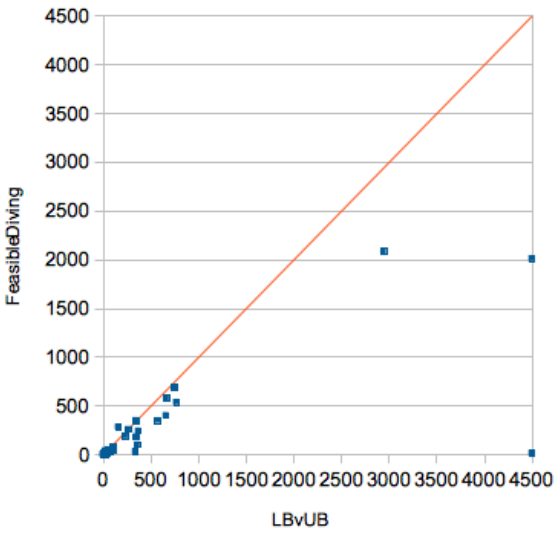

FIGURE 3 – `LBvUB` et `FD`. Les coordonnées de chaque point représentent les temps CPU (en secondes) utilisés par les compétiteurs.

`Diving`, est appelée à chaque nœud d'un algorithme B&B en mailleur d'abord en utilisant une variante du critère LB. La procédure `FeasibleDiving` plonge de manière gloutonne ne gardant qu'un nœud à chaque itération. Chaque nœud traité par cette procédure est contracté et un point réalisable y est cherché.

Ces deux stratégies donnent de meilleurs résultats que l'algorithme B&B standard en meilleur d'abord. `FeasibleDiving` permet de résoudre deux instances supplémentaires. Nous pensons que cette stratégie bénéficie d'une synergie avec les procédures de recherche de point réalisables disponibles dans `IbexOpt` et utilisées par la procédure `FeasibleDiving`.

Références

[1] I. Araya, G. Trombettoni, and B. Neveu. A Contractor Based on Convex Interval Taylor. In *Proc. CPAIOR*, volume 7298 of *LNCS*, pages 1–16. Springer, 2012.

[2] I. Araya, G. Trombettoni, B. Neveu, and G. Chabert. Upper Bounding in Inner Regions for Global Optimization under Inequality Constraints. *J. Global Optimization (JOGO)*, 60(2) :145–164, 2014.

[3] F. Benhamou, F. Goualard, L. Granvilliers, and J.-F. Puget. Revising Hull and Box Consistency. In *Proc. ICLP*, volume 5649 of *LNCS*, pages 230–244. Springer, 1999.

[4] R.E. Bixby and E. Rothberg. Progress in Computational Mixed Integer Programming – A Look Back from the Other Side of the Tipping Point. *Annals of Operations Research*, 149 :37–41, 2007.

[5] P. Bonami, M. Kilink, and J. Linderoth. Algorithms and Software for Convex Mixed Integer Nonlinear Programs. Technical Report 1664, U. Wisconsin, 2009.

[6] L.G. Casado, J.A. Martinez, and I. Garcia. Experiments with a New Selection Criterion in a Fast Interval Optimization Algorithm. *Journal of Global Optimization*, 19 :247–264, 2001.

[7] G. Chabert. Interval-Based EXplorer, 2015. `www.ibex-lib.org`.

[8] G. Chabert and L. Jaulin. Contractor Programming. *Artificial Intelligence*, 173 :1079–1100, 2009.

[9] T. Csendes. New Subinterval Selection Criteria for Interval Global Optimization. *Journal of Global Optimization*, 19 :307–327, 2001.

[10] T. Csendes and D. Ratz. Subdivision Direction Selection in Interval Methods for Global Optimization. *SIAM Journal on Numerical Analysis*, 34(3), 1997.

Gain	gain > 5	gain $[2, 5]$	gain $[1.2, 2]$	gain $[1.05, 1.2]$	equiv. $[0.95, 1.05]$	perte $[0.8, 0.95]$	perte $[0.5, 0.8]$	perte < 0.5
#instances	2	10	24	12	25	5	5	1

TABLE 5 – Détail des gains/pertes de FD par rapport à `LBvUB`. Les gains sont exprimés par $\frac{time(LBvUB)}{time(FD)}$. Les valeurs sont les nombres d'instances ayant un gain dans un intervalle donné.

[11] A. Felner, S. Kraus, and R. E. Korf. KBFS : K-Best-First Search. *Annals of Mathematics and Artificial Intelligence*, 39, 2003.

[12] R.B. Kearfott and M. Novoa III. INTBIS, a Portable Interval Newton/Bisection Package. *ACM Trans. on Mathematical Software*, 16(2) :152–157, 1990.

[13] L. Kocsis and C. Szepesvari. Bandit based Monte-Carlo Planning. In *Proc. ECML*, volume 4212 of *LNCS*, pages 282–293. Springer, 2006.

[14] M.C. Markot, J. Fernandez, L.G. Casado, and T. Csendes. New Interval Methods for Constrained Global Optimization. *Mathematical Programming*, 106 :287–318, 2006.

[15] F. Messine. *Méthodes d'optimisation globale basées sur l'analyse d'intervalle pour la résolution des problèmes avec contraintes*. PhD thesis, LIMA-IRIT-ENSEEIHT-INPT, Toulouse, 1997.

[16] F. Messine and J.-L. Laganouelle. Enclosure Methods for Multivariate Differentiable Functions and Application to Global Optimization. *Journal of Universal Computer Science*, 4(6) :589–603, 1998.

[17] B. Neveu, G. Trombettoni, and I. Araya. Adaptive Constructive Interval Disjunction : Algorithms and Experiments. *Constraints Journal*, DOI : 10.1007/s10601-015-9180-3 :Accepted for publication, 2015.

[18] J. Ninin and F. Messine. A Metaheuristic Methodology Based on the Limitation of the Memory of Interval Branch and Bound Algorithms. *Journal of Global Optimization*, 50 :629–644, 2011.

[19] J. Ninin, F. Messine, and P. Hansen. A Reliable Affine Relaxation Method for Global Optimization. *4OR-Quaterly Journal of Operations Research*, 2014. Accepted for publication. DOI : 10.1007/s10288-014-0269-0.

[20] A. Sabharwal, H. Samulowitz, and C. Reddy. Guiding Combinatorial Optimization with UCT. In *Proc. CPAIOR*, volume 7298 of *LNCS*, pages 356–361. Springer, 2012.

[21] M. Tawarmalani and N. V. Sahinidis. Global Optimization of Mixed-Integer Nonlinear Programs : A Theoretical and Computational Study. *Mathematical Programming*, 99(3) :563–591, 2004.

[22] G. Trombettoni, I. Araya, B. Neveu, and G. Chabert. Inner Regions and Interval Linearizations for Global Optimization. In *Proc. AAAI*, pages 99–104, 2011.

[23] G. Trombettoni and G. Chabert. Constructive Interval Disjunction. In *Proc. CP*, volume 4741 of *LNCS*, pages 635–650. Springer, 2007.

A Liste des 84 instances d'optimisation globale sous contraintes testées

Nom	Br.	Name	Br.	Name	Br.	Nom	Br.
ex2_1_9	ssr	ex8_2_1	ssa	linear	ssr	hs088	lf
ex3_1_1	ssr	ex8_4_4	ssr	meanvar	ssr	hs093	ssr
ex5_3_2	ssr	ex8_4_5	lf	process	ssr	hs100	ssr
ex5_4_3	ssr	ex8_4_6	ssr	ramsey	lf	hs103	ssr
ex5_4_4	ssa	ex8_5_1	ssr	sambal	rr	hs104	lf
ex6_1_1	ssr	ex8_5_2	ssr	srcpm	sm	hs106	lf
ex6_1_3	ssr	ex8_5_6	ssr	avgasa	ssr	hs109	ssr
ex6_1_4	ssr	ex14_1_2	ssr	avgasb	ssr	hs113	lf
ex6_2_6	ssr	ex14_1_6	ssr	batch	ssa	hs114	rr
ex6_2_8	ssr	ex14_1_7	ssr	dipigri	ssr	hs117	ssa
ex6_2_9	ssr	ex14_2_1	ssr	disc2	ssr	hs119	ssa
ex6_2_10	ssr	ex14_2_3	ssr	dixchlng	lf	makela3	ssr
ex6_2_11	ssr	ex14_2_7	ssr	dualc1	ssr	matrix2	lf
ex6_2_12	ssr	alkyl (rr)	lf	dualc2	ssr	mistake	ssa
ex7_2_3	ssr	bearing	ssr	dualc5	ssr	odfits	ssr
ex7_2_4	lf	hhfair	ssr	genhs28	lf	optprloc	ssr
ex7_2_8	lf	himmel16	ssr	haifas	ssr	pentagon	ssr
ex7_2_9	lf	house	ssr	haldmads	lf	polak5	ssr
ex7_3_4	ssr	hydro	ssr	himmelbk	lf	robot	lf
ex7_3_5	ssr	immun	rr	hs056	lf	concon	ssr
ex8_1_8	ssr	launch	ssr	hs087	ssr	mconcon	ssr

TABLE 6 – Banc d'essai testé. Les systèmes ont été sélectionnés dans le banc d'essai Coconut (séries 1 et 2) suivant le protocole décrit à la partie 5. La meilleure stratégie de branchement (colonne Br.) a été choisie pour chaque système, la même pour toutes les méthodes, parmi : plus grand intervalle (lf), tour de rôle (rr), Smear max (sm), Smear sum absolue (ssa) (voir [12]) et Smear sum relative (ssr) (voir [22]).